Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Analyzing Ship Wakes in Synthetic Aperture Radar Images

Development of the "RedBox" Application

A.J. Higginson

Defence R&D Canada - Ottawa
TECHNICAL MEMORANDUM
DRDC Ottawa TM 2002-157
December 2002

Canada

Analyzing Ship Wakes in Synthetic Aperture Radar Images

Development of the "RedBox" Application

A. J. Higginson
Space Systems and Technology Section

Defence R&D Canada – Ottawa

TECHNICAL MEMORANDUM

DRDC Ottawa TM 2002-157

December 2002

© Her Majesty the Queen as represented by the Minister of National Defence, 2002

© Sa majesté la reine, représentée par le ministre de la Défense nationale, 2002

106-03711/

Abstract

This report summarizes the development of an application for analyzing ships and ship wakes found in satellite Synthetic Aperture Radar (SAR) imagery. The need for such an application is identified and a previously developed prototype application is examined briefly.

The new application, known as RedBox, offers semi-automated ship wake detection. Ship wakes are detected by identifying significant line-tendencies in the SAR image with the help of a Radon transformation. Image processing techniques such as high-pass filtering using the Fast Fourier Transform are available to enhance the detection process.

Current shortcomings of the application are mentioned and future plans for development are discussed. Relevant C++ source code is included for image processing and transformation algorithms.

Résumé

Ce document résume le développement d'un logiciel pour analyser les bateaux et les sillages de bateau trouvés dans les images satellite de Radar à Synthèse d'Ouverture (RSO). Le besoin d'une telle application est identifié et un logiciel de prototype est brièvement examiné.

La nouvelle application, connue sous le nom de RedBox, offre une détection semi-automatisée de sillage de bateau. Des sillages sont détectés en identifiant les tendances de trait de ligne significatives dans l'image de RSO avec l'aide d'une transformé de Radon. Les techniques de traitement d'image telles que le filtrage passe-haut en utilisant la transformée de Fourier Rapide sont disponibles pour assister à la détection.

Quelques imperfections actuelles de l'application sont mentionnées et des plans pour le développement futur sont discutés. Le code source de C++ est inclus pour les algorithmes de traitement et de transformation d'image.

This page intentionally left blank.

Executive summary

Space-borne SAR imaging equipment can be used to provide valuable data for naval tracking and surveillance programs. Detailed analysis of a ship or ship wake in a SAR image could reveal important information about that particular vessel's size and heading. Also, since the Doppler effect affects the location of the ship on the radar image it is possible to calculate the velocity of the ship mathematically, given certain radar parameters and the displacement of the ship image from the wake apex. Detection of sub-surface vessels is also possible as they may have a wake that is visible in a SAR image.

A software tool for wake analysis is currently being developed. The application, named "RedBox", is an extension and redesign of a prototype application developed during the summer of 2002. Features of the RedBox application include:

- Magnification of selected regions of an image. Manual wake analysis requires close inspection of certain key areas of an image.
- Image enhancement using algorithms such as the high-pass filter. Low frequency image data can interfere with the accuracy of wake-detection algorithms.
- Detection of significant line tendencies. Ship wakes are often visible as a series of intersecting straight lines. Use of the Radon transform is an integral part of the line-detection procedure.

The application makes use of several mathematical transformations and image processing algorithms. High-pass filtering is accomplished by applying a Fast Fourier Transform to the image and then masking out the centre (low frequency) region of the transformed image. The filtered result appears after performing a reverse FFT. Line detection is accomplished using the Radon Transform, which converts a two-dimensional image into a domain of line parameters, with strong line tendencies shown as very bright or very dark points in the Radon domain. This reduces the task of detecting lines in the SAR image to finding the bright and dark points in the Radon domain. Depending on the characteristics of the image, the Radon transform will sometimes identify lines that do not belong to a ship wake. A Wiener-type filter is used to minimize these 'false alarms' by comparing the pixels surrounding a Radon point to an ideal wake-arm point model. If there is a large correlation the magnitude of the point is increased and vice versa. C++ source code for all algorithms is included as an annex to this report.

Continued development of the software will be necessary as the target of a fully automatic wake analysis tool has not yet been reached. Also, significant improvement can be made to the user interface of the tool in terms of functionality and reliability.

Higginson, A.J. 2002. Analyzing Ship Wakes in Synthetic Aperture Radar Images – Development of the "RedBox" Application. DRDC Ottawa TM 2002-157. Defence R&D Canada - Ottawa.

Sommaire

Un satellite qui produit des images RSO peut être employé pour fournir des données de qualité aux programmes de cheminement et de surveillance navale. L'analyse détaillée d'un bateau ou d'un sillage de bateau dans une image de RSO peut indiquer des informations importantes sur la taille et l'entête de ce navire particulier. Aussi, puisque l'effet Doppler affecte l'emplacement du bateau sur l'image radar, il est possible de calculer la vitesse du bateau, en étant donné certains paramètres du radar et la distance entre le bateau et le sillage sur l'image. La détection des navires sous-marins est également possible car elles peuvent avoir un sillage qui est visible dans une image de RSO.

Un outil logiciel pour l'analyse de sillages est en train d'être développé. Ce logiciel, nommée " RedBox ", est une évolution et une extension d'un logiciel de prototype développé durant l'été de 2002. Les dispositifs du logiciel RedBox incluent:

- Agrandissement de régions sélectionnées d'une image. L'analyse manuelle de sillage exige l'inspection minutieuse de certaines régions principales d'une image.
- Amélioration de l'image en utilisant des algorithmes tels que le filtre passe-haut. Les traits de basse fréquence image peuvent fausser la détection de sillages.
- Détection des tendances de lignes significatives. Les sillages de bateau sont souvent visibles comme série entrecroisant de lignes droites. L'utilisation de la transformé de Radon est une partie intégrale du procédé de détection de ligne.

Le logiciel RedBox se sert de plusieurs transformations et algorithmes mathématiques pour traitement d'image. Le filtrage passe-haut est accompli en appliquant une transformée de Fourier Rapide à l'image et ensuite en masquant la région centrale (basse fréquence) de l'image. Le résultat filtré apparaît après avoir exécuté la transformée de Fourier renversé. La détection des traits linéaires est accomplie en utilisant la transformée de Radon, qui prend une image bidimensionnel et la transforme en un domaine de paramètres linéaires, avec des tendances linéaires fortes indiquées dans le domaine Radon avec un point lumineux ou très foncé. La tâche de détection de lignes dans l'image de RSO est réduite à trouver les points lumineux et foncés dans le domaine de radon. Selon les caractéristiques de l'image, la transformée de Radon identifiera parfois des lignes qui n'appartiennent pas à un sillage de bateau. Un filtre Wiener est utilisé pour réduire au minimum ces 'alarmes fausses' en comparant les pixels entourant un point de radon à un modèle de point de sillage idéal. S'il y a une grande corrélation, l'importance du point est augmentée, et vice versa. Le code source de C++ pour tous les algorithmes est inclus comme annexe de ce document. Le développement continué du logiciel sera nécessaire car la cible d'un outil d'analyse de sillage entièrement automatique n'est pas encore réalisé. L'amélioration de l'interface utilisateur de l'outil en termes de fonctionnalité et fiabilité est aussi possible.

Higginson, A.J. 2002. Analyzing Ship Wakes in Synthetic Aperture Radar Images – Development of the "RedBox" Application. RDDC Ottawa TM 2002-157. R & D pour la défense Canada - Ottawa

Table of contents

Abstract i

Executive summary iii

Sommaire iv

Table of contents v

List of figures viii

Acknowledgements x

1. Introduction 1
   - 1.1 SAR Imagery and Monitoring Naval Activity 1
   - 1.2 Development of a Software Tool for Wake Analysis 1

2. The Beginning – RedBox v1.0 2
   - 2.1 History of Version 1.0 2
   - 2.2 Features 2
     - 2.2.1 Conversion to 24-Bit Windows Bitmap Format (DIB) 2
     - 2.2.2 Magnification Tool 2

3. Current Status – RedBox v3.0 3
   - 3.1 A New Direction for RedBox 3
   - 3.2 Features 3
     - 3.2.1 User Interface 3
     - 3.2.2 Conversion to 24-Bit Windows Bitmap Format (DIB) 3
     - 3.2.3 Magnification Tool – The Zoom Window 4
     - 3.2.4 Image Filtering and Optimization 6
     - 3.2.5 Automatic Wake Detection 7
     - 3.2.6 Line Manipulation 8
     - 3.2.7 Display Options 8
   - 3.3 Algorithms 11
     - 3.3.1 Radon Transform 11

3.3.2 High-Pass Filtering 13
3.3.3 Wiener Filtering 17

4. Future Developments for Software 18
4.1 Ship Velocity Calculation 18
4.2 Refinement of Wake Detection Algorithm 18
4.3 DLL Packaging 18
4.4 Enhancing Interface Functionality and Aesthetics 18

5. Observations 20

6. Conclusions 21

7. References 22

Annex A – Screenshots 23

Annex B – Microsoft Windows 24-Bit Bitmap file format 26
File Header 27
Format 27
Members 27
Data Header 28
Format 28
Members 28
Data 30

Annex C – Application Code 31
_CMPLX data structure 31
Bitmap Format Conversion 31
Fourier Transform 33
High-Pass Filter 36
Radon Transform 37
Wiener Filter 44
Line Detection (histogram) 53
Line Detection (standard deviation) 55
Drawing Radon lines 57

Generating the Complex Data Array from a Bitmap 60
Generating a Bitmap from the Complex Data Array (real part only) 61
Generating a Bitmap from the Complex Data Array (real and imaginary parts) 63

Bibliography 65

List of figures

Figure 1. Converting Bitmaps to 24-Bit Windows Format 4

Figure 2. A red box outlines the area being magnified..... 4

Figure 3. The magnified image is visible in the Zoom window 5

Figure 4. Selecting a different magnification factor..... 5

Figure 5. Closing the Zoom window from the Window menu..... 6

Figure 6. High-Pass Filter Parameters 6

Figure 7. Line detection algorithms are invoked by clicking either "Histogram Detect" or "SD Detect". Detected lines are displayed in the list box. 7

Figure 8. The Transform Window Display Property Page. 9

Figure 9. Transform Zoom set at 100 percent. View size matches size of selected region. 10

Figure 10. Transform Zoom set at 300 percent. View size is three times larger than selected region..... 10

Figure 11. Line in x-y domain corresponding to Radon point at (ρ,θ) 11

Figure 12. For a square transform region, the maximum value of $\rho \max = \left(\frac{width}{2}\right) \times \sqrt{2}$. 12

Figure 13. Diagonal Line – original image on left, radon transformed image on right. The coordinates of the bright white point correspond exactly to the parameters of the line.... 12

Figure 14. SAR ship wake – original on left, radon image on right. The bright points correspond to the wake arms. Notice scalloping effect around the edges of the radon image. 12

Figure 15. The centred Fourier Domain 13

Figure 16. Source image for 2D FFT algorithm example..... 14

Figure 17. Result after applying 2D FFT algorithm to the source image. The image on the left is the result of transforming the raw image data while the image on the right is the result of transforming the image with centring matrix applied. 14

Figure 18. The weighting function describes a sinusoid along the radius of the weighting region between two user-defined radii. 15

Figure 19. The three regions of the image filter. Image data is unaltered outside the outer radius. 15

Figure 20. Result after applying high-pass weighting function to centred image. 15

Figure 21. The filtered result after applying a reverse FFT and removing the centring matrix.16

Figure 22. SAR ship wake – Source image on left, High-pass filtered image on right. 16

Figure 23. Opening a SAR bitmap image for processing 23

Figure 24. The Transform view with the transform area on the original bitmap surrounded by a green box. 24

Figure 25. Performing a Radon transform – a progress bar appears at the bottom of the window pane. 24

Figure 26. The Radon transform complete – two points are selected, one on each wake arm. Notice the associated lines drawn in the transform area of the original image. The wake axis has also been calculated and is represented by a white line on the image. 25

Figure 27. Both the transform and zoom views are shown. It is possible to work with both views at once. The zoom view shows a magnified image of the area bounded by the red box. 25

Acknowledgements

Many thanks to my supervisor and mentor Jake Tunaley for sharing his knowledge and providing praise and criticism throughout the development process.

1. Introduction

1.1 SAR Imagery and Monitoring Naval Activity

Synthetic Aperture Radar (SAR) images generated from space-borne radar satellites such as RADARSAT often show clear evidence of the presence of ships and ship wakes on the sea. Detailed information about the ships that are visible in these satellite images is of significant interest to anyone wishing to track and/or classify naval traffic. Unfortunately, it is often difficult or impossible to determine the heading and velocity of a vessel upon visual inspection of a SAR image. This is because the image of the ship and its orientation are often distorted by ship motion.

A satellite incorporating a steering mechanism for a SAR beam would allow the system to measure the position of a vessel at more than one point in time, so that the velocity could be deduced from its displacement in the image plane. However, beam steering and onboard processing to implement this adds considerably to the cost of the satellite.

Fortunately, there is another alternative. The analysis of ship wakes in SAR imagery using basic image-processing techniques and specialized algorithms can provide significant information about a wake's associated vessel, including the approximate size and heading of the ship. The velocity of the vessel can be estimated by measuring the displacement of the ship relative to the apex of the wake. This is because the wake is more or less stationary on the surface of the water while the radar returns from the ship are Doppler-shifted. The Doppler shift displaces the position of the ship in the final image. Sub-surface vessels that are not visible in SAR images could also possibly be detected based on the size and shape of their wake. With the proper analysis tools it should be possible to obtain reasonably accurate data.

1.2 Development of a Software Tool for Wake Analysis

This report outlines the development of a software tool to assist in extracting ship and ship wake information from satellite SAR imagery. The application is being developed under Microsoft Visual C++ 6.0 and uses the Microsoft Foundation Classes (MFC) framework. Visual C++ was selected as the development language for several reasons. The power and speed of C++ is desirable for calculation-intensive algorithms such as the Radon transform and the Wiener filter. The MFC framework provides a familiar user interface that will drastically reduce the learning curve for first-time users of the software. Also, the widespread use of C++ throughout the world allows the re-use of important algorithms in other simulation and analysis applications without the need for intensive code translation or debugging.

2. The Beginning – RedBox v1.0

2.1 History of Version 1.0

The first steps toward developing a comprehensive tool for wake image analysis were taken during the summer of 2002. The result of these efforts, dubbed RedBox, was a simplistic dialog-based prototype application with limited functionality. A brief overview of the application's features will be provided here – for a full report, refer to Satellite Radar Detection of Ship Wakes (A.T.E Williams, DRDC Ottawa TN 2002-079).

2.2 Features

2.2.1 Conversion to 24-Bit Windows Bitmap Format (DIB)

The current methods of extracting images from SAR image tape and CD-ROM media involve using a software package named ENVI, running on a Silicon Graphics machine. ENVI has the ability to read SAR data and save images as bitmaps. However, the bitmap format used by the ENVI application is not compatible with the Microsoft Windows Bitmap Standard. RedBox v1.0 has the ability to convert ENVI-standard bitmapped images to Windows Standard 24-bit bitmaps so that they can be viewed with any generic windows bitmap viewer.

2.2.2 Magnification Tool

In order to determine the velocity of a vessel from it's wake and position, it is necessary to measure the displacement of the ship from the apex of the wake. For this reason, the RedBox application has the ability to magnify regions of an image by a factor of 10. This feature is used to facilitate manual measurement of the distance, in pixels, between the ship and the axis of the wake. The magnification region is selected by using the mouse to define two opposite corners of a box (drawn in red – hence the name RedBox). Once the region has been defined, a right click displays the magnified view.

3. Current Status – RedBox v3.0

3.1 A New Direction for RedBox

At the beginning of September 2002 the primary developer of RedBox was replaced by another co-op student from Carleton University. This was the advent of a number of significant changes for RedBox. The focus of the application shifted from being a tool to assist manual calculation toward being a full-featured automatic wake detection and analysis program. Image processing routines involving Fourier and Radon transforms were employed and automatic wake detection algorithms were added. Other features, including ship velocity calculation, are under development.

3.2 Features

3.2.1 User Interface

The dialog-based approach of version 1.0 did not provide the flexibility necessary to implement some of the new features therefore a decision was made to build the current version of RedBox on the MFC Multiple Document Interface (MDI) framework. The MDI uses a document-view architecture, which "provides a framework for managing an application's data and user interface" [1]. The radar image is considered to be the *document* (data) and the user can open one or more *views* (e.g. Zoom window, Transform window, etc…) on that document.

When an image document is opened in the application, the default view is displayed. Additional views such as the Zoom view and the Transform view are accessible in the menu bar at the top of the application. The application provides a standard Windows look and feel in order to avoid a lengthy familiarization period for new users.

3.2.2 Conversion to 24-Bit Windows Bitmap Format (DIB)

The headers in the bitmap files produced by the ENVI program are not in a standard format and the files cannot be displayed in Microsoft Paint. As in version 1.0, the current version of the RedBox application is capable of converting most bitmaps to a standard 24-bit Windows Bitmap format. This feature is accessible from the File menu at any time during the execution of the program.

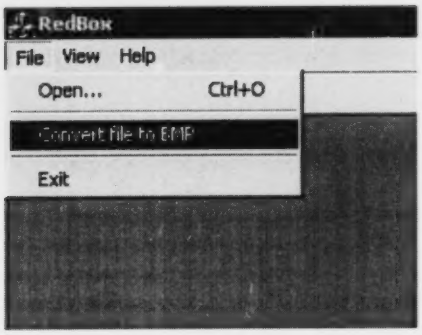

Figure 1. *Converting Bitmaps to 24-Bit Windows Format*

When selected, a dialog box prompts the user to select the file to convert. Upon successful conversion, a message box is displayed.

3.2.3 Magnification Tool – The Zoom Window

The Zoom Window allows the user to select a square region of the original image that they would like to see magnified in a separate window of the application. When enabled, the zoom window appears in the top right corner of the application, and a red box surrounding the zoom region appears in the main window.

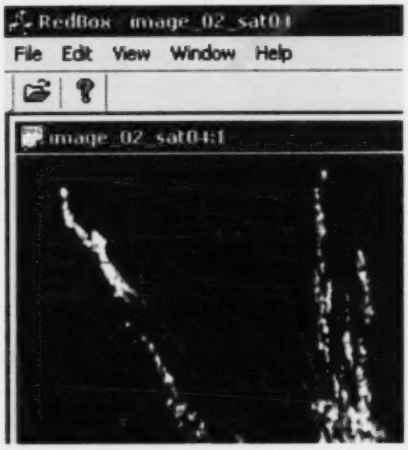

Figure 2. *A red box outlines the area being magnified*

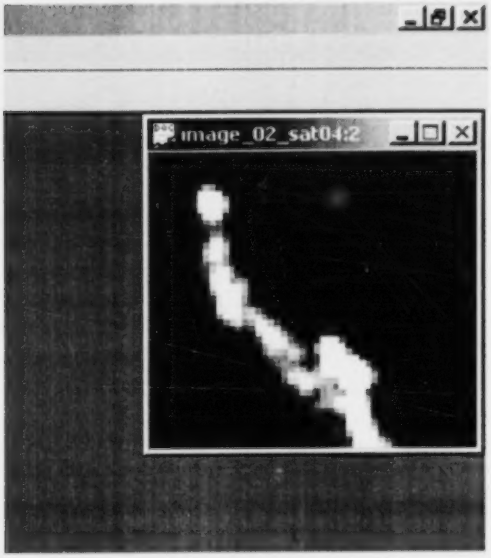

Figure 3. *The magnified image is visible in the Zoom window*

Magnification factors start at 1x (no zoom) and increase in powers of two. There is no upper limit but it is often unnecessary to exceed 32x zoom, as the most useful magnifications factors are often those from 2x to 16x. These common factors can be selected directly from the pop-up menu that appears when right-clicking either the zoom window or the main window when the zoom window is visible.

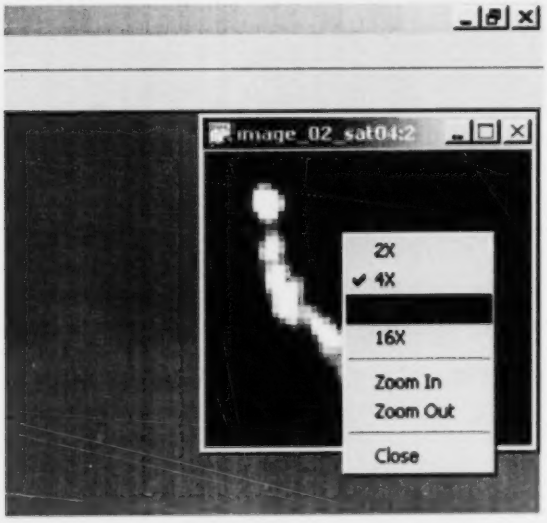

Figure 4. *Selecting a different magnification factor*

The zoom region can be moved by clicking inside the red box and dragging it to the desired location. To close the zoom window, select 'close' from the popup menu, or toggle the Zoom Window option from the Window menu.

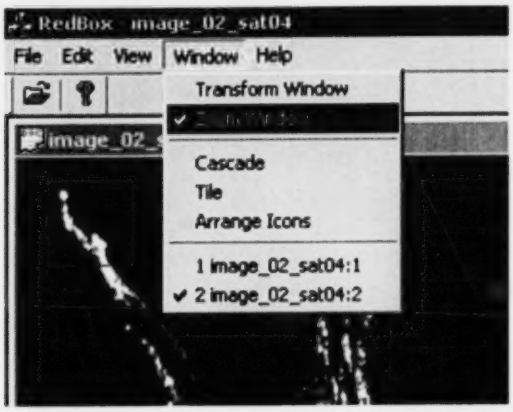

Figure 5. *Closing the Zoom window from the Window menu*

3.2.4 Image Filtering and Optimization

The quality and characteristics of SAR images vary greatly between different satellites due to the type and quality of the radar device in question. For this reason, it is often necessary to pre-process an image before attempting to detect wakes and wake features. Gradual variations in background intensity will negatively affect the line-detection algorithms therefore the RedBox application features high-pass filtering capabilities to remove this low-frequency phenomenon. The RedBox high-pass filter has customizable width and ramp parameters for fine-tuning in cases where the default filtering values are not well suited to the image being processed.

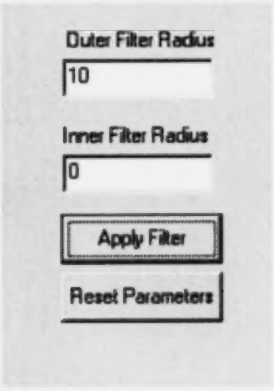

Figure 6. *High-Pass Filter Parameters*

The RedBox application also features a Wiener-style filter for improving the accuracy of the line-detection algorithms. The Wiener filter, when applied after a Radon transform, enhances areas of the image that show a strong wake-tendency, and suppresses areas of the image that most likely represent false alarms.

Specific details regarding these filters and transforms can be found in Section 3.3 – Algorithms.

3.2.5 Automatic Wake Detection

The major feature of the RedBox application is its ability to detect ship wakes in SAR imagery. Using a combination of filters, transforms and sorting algorithms, RedBox users can locate and select areas of an image that show strong wake characteristics. Since a typical wake structure usually consists of two or more straight lines, the RedBox application is designed to scan the selected transformation region for straight-line tendencies. Currently, the user has a choice between two methods for peak detection – a histogram-style threshold, or a threshold based on the standard deviation. Both methods are quite effective at detecting wake arms in an image, however each method does occasionally produce false alarms: lines which do not belong to the ship wake but which are still selected by the detection algorithm. Manually adjusting the upper and lower detection bounds appropriately can minimize the number of false alarms. Once detection is complete, the algorithms create and display a list of all lines in the image that meet the detection criteria.

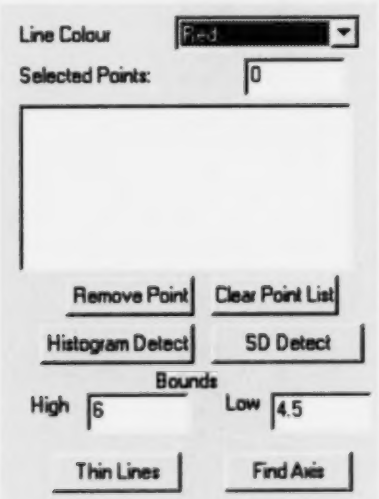

Figure 7. *Line detection algorithms are invoked by clicking either "Histogram Detect" or "SD Detect". Detected lines are displayed in the list box.*

3.2.6 Line Manipulation

Occasionally the automatic detection routines select too few or too many lines. In such cases, it is possible to add or remove lines manually. In the case of too many lines, the user can review the list of selected lines and remove unwanted lines simply by selecting the line and clicking the "Remove Point" button. When a line is selected in the list, the corresponding line on the image is drawn with a thick pen so that the user is certain which line will be affected. To remove all lines from the list, click "Clear Point List". For severe cases when the automatic detector selects an extremely large number of lines, there is a thinning routine that chooses one or two central points from a cluster of detected points and deletes the rest of the cluster. This makes the selected points list a great deal smaller. The line-thinning feature is still under development and will occasionally give false results but even at this stage it can be quite useful at times.

In the case where there are too few lines, additional lines can be selected manually by clicking the left mouse button on the corresponding location in the transform view. Every click adds a point to the detection list.

It is possible to specify the line colour of your detected lines – simply choose the desired colour from the drop-down box before detecting lines. It is also possible (and often quite useful) to assign different line colours to different lines. When performing an automatic detection, the software automatically assigns the current colour to lines that meet the upper criterion, and lines between the upper and lower criteria are assigned the next colour in the list.

When an acceptable number of lines are selected, it is possible to calculate the wake axis (centreline) for this collection by clicking the "Find Axis" button. The axis is determined in the Radon domain by averaging the coordinates of the selected points separately. The resulting line is displayed on the image. However, a large imbalance between the numbers of selected lines on each wake arm will produce skewed results and a meaningless wake axis line.

3.2.7 Display Options

Often it is desirable to resize the transform region so that the target wake fills the transform region. RedBox allows the user to resize the transform region by choosing a preset value from the Display Parameters tab, or by dragging the corner of the transform region to the desired size. Dragging the transform region corner increases or decreases the size of the transform region in increments of two pixels along each axis.

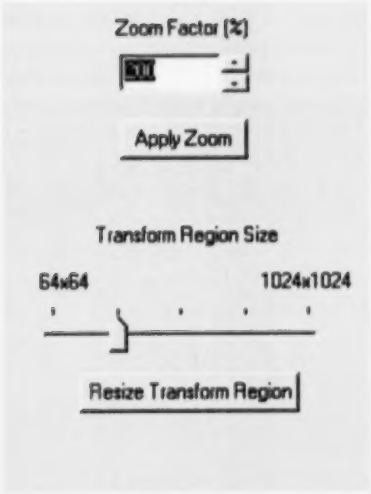

Figure 8. *The Transform Window Display Property Page.*

To set the transform region size with the display property page, simply move the slider to the desired size and click "Resize Transform Region". All views will be updated accordingly.

For extremely large or extremely small transform regions the image displayed in the Transform window will most likely be sized inconveniently, showing either a tiny picture which offers no detail or displaying a huge picture requiring the user to scroll a great distance from side to side. RedBox offers a Transform Zoom feature that allows the user to adjust the display in the transform window to a convenient size. Simply specify the desired size (as a percentage of the source image size) and click "Apply Zoom".

Figure 9. *Transform Zoom set at 100 percent. View size matches size of selected region.*

Figure 10. *Transform Zoom set at 300 percent. View size is three times larger than selected region.*

3.3 Algorithms

The RedBox application makes use of several algorithms for data and image manipulation. These algorithms are described below, and the C++ code to implement them is included in Annex C – Application Code.

3.3.1 Radon Transform

The Radon Transform is used to "transform two dimensional images with lines into a domain of possible line parameters, where each line in the image will give a peak positioned at the corresponding line parameters"[2]. Performing a Radon transform on the image allows the software to search for very bright or very dark points in the Radon transform which represent very strong bright or dark lines in the image. Since most ship wakes in SAR images appear as a combination of bright and dark lines, this technique is very useful for wake detection.

The Radon Domain uses a (ρ,θ) coordinate system. Each point corresponds to a line perpendicular to a vector from the origin with a length of ρ units, at an angle of θ radians to the positive x-axis.

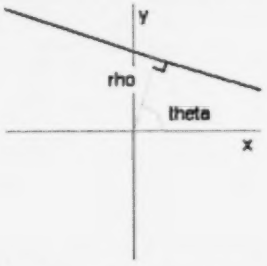

Figure 11. *Line in x-y domain corresponding to Radon point at* (ρ,θ)

In the RedBox application, the horizontal axis of the Radon domain is the displacement, ρ. The angle, θ, is centred along the vertical axis at a value of $\frac{\pi}{2}$, with zero radians at the top, and π radians at the bottom.

Before beginning the transform, the Radon algorithm in RedBox determines the maximum dimensions of the Radon image based on the dimensions of the original image. The transformed image retains the original image height, and the change in θ for one pixel is calculated so that the y-axis ranges in value from 0 to π. The width of the Radon image is a factor of $\sqrt{2}$ times larger than the original image, since the maximum ρ value that will be encountered is equal to the length of the diagonal, ρ max.

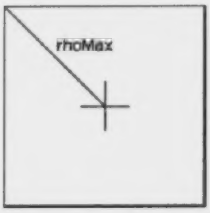

Figure 12. *For a square transform region, the maximum value of* $\rho \max = \left(\frac{width}{2}\right) \times \sqrt{2}$

Having calculated these quantities, the algorithm then iterates over θ from 0 to π and then for each angle iterates over ρ from $-\rho \max$ to $+\rho \max$. For all iterations, the mean of all pixels along the specified line is calculated and is stored in the Radon domain as the value for that pair of line-parameters. For combinations of ρ and θ beyond the limits of the image, a magnitude equal to the image mean is assigned in the Radon domain. This occasionally appears as a scalloped region on the right and left edges of the radon domain.

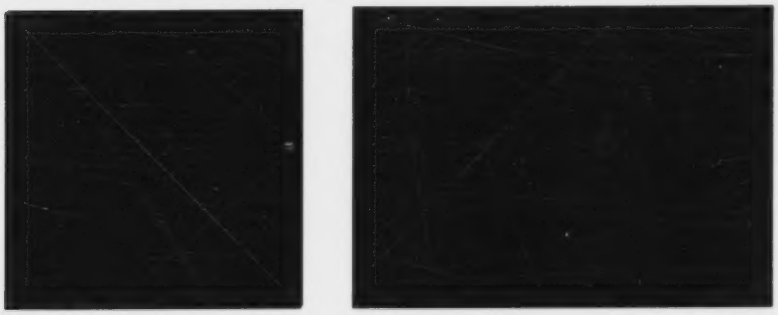

Figure 13. *Diagonal Line – original image on left, radon transformed image on right. The coordinates of the bright white point correspond exactly to the parameters of the line.*

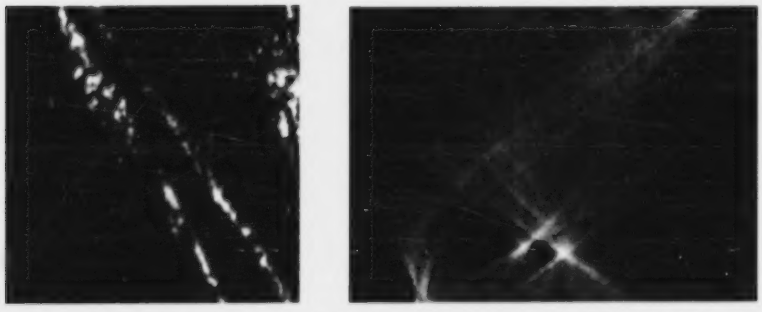

Figure 14. *SAR ship wake – original on left, radon image on right. The bright points correspond to the wake arms. Notice scalloping effect around the edges of the radon image.*

3.3.2 High-Pass Filtering

In this context, a high-pass filter is a filter that removes low frequency signals from an image while letting higher frequencies pass through unchanged. The RedBox high-pass filtering algorithm makes use of a two-dimensional Fourier transform and a weighting function in order to mask low frequency variation in an image while preserving the higher frequency data. Low frequency variations in an image will severely affect the reliability of the wake detection algorithm used in the software, causing it to generate an increased number of false alarms.

Before beginning the filtering process, a centring matrix is applied to the image. The centring matrix for a two-dimensional Fourier transform is a matrix of positive and negative ones, as described by:

$$n_{ij} = (-1)^{i+j}$$

This preparation of the data ensures that the image in the Fourier domain will be centred, with zero frequency in the middle of the image and frequency increasing or decreasing appropriately along each axis.

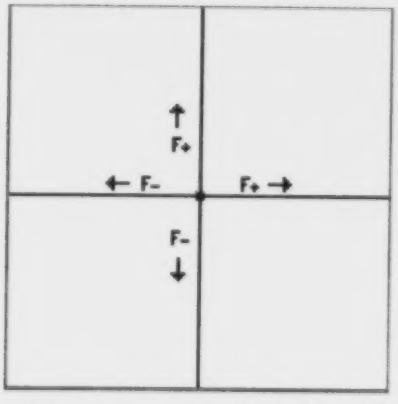

Figure 15. *The centred Fourier Domain*

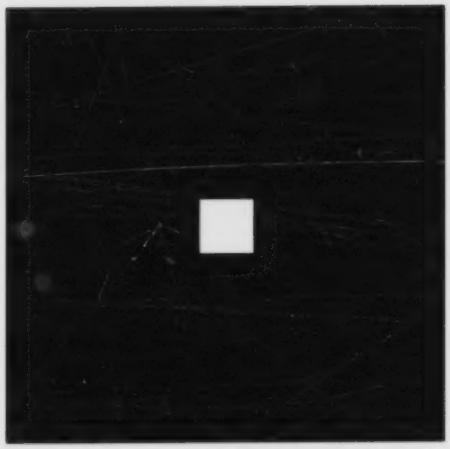

Figure 16. *Source image for 2D FFT algorithm example*

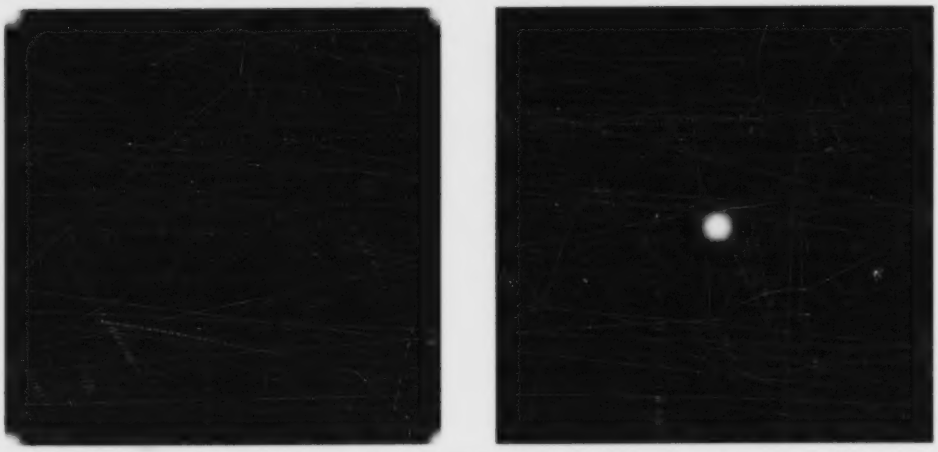

Figure 17. *Result after applying 2D FFT algorithm to the source image. The image on the left is the result of transforming the raw image data while the image on the right is the result of transforming the image with centring matrix applied.*

Once the Fourier domain is centred it is quite simple to remove all low frequencies below a specified boundary by applying a circular weighting function to the centre of the image. Since sharp edges in the image will produce a ringing effect in the Fourier transform results, the weighting region was chosen to be a sinusoidal gradient that produces a relatively smooth transition from full signal level to zero. The filtering function accepts two parameters: an outer radius representing the frequency at which the weighting will begin, and an inner radius representing the frequency at which the gradient weighting reaches zero. All points between the lower radius and the origin are weighted to a value of zero.

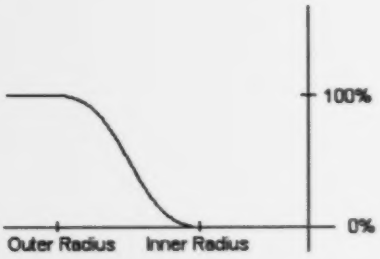

Figure 18. *The weighting function describes a sinusoid along the radius of the weighting region between two user-defined radii.*

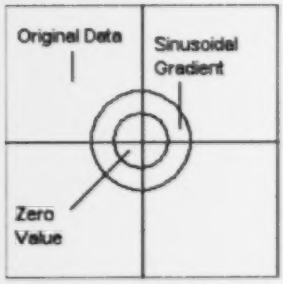

Figure 19. *The three regions of the image filter. Image data is unaltered outside the outer radius.*

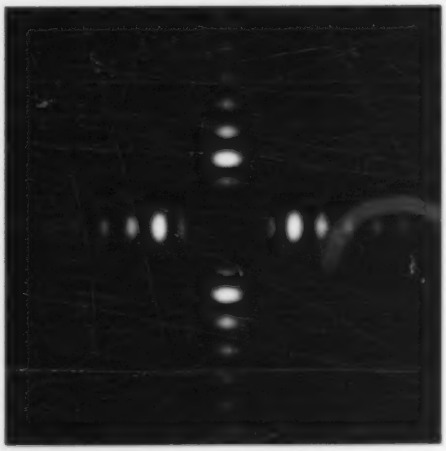

Figure 20. *Result after applying high-pass weighting function to centred image.*

Once the weighting function has been applied, a reverse Fourier transform is applied to the image and the image is multiplied once again by the centring matrix, thereby cancelling the effect of the first centring matrix.

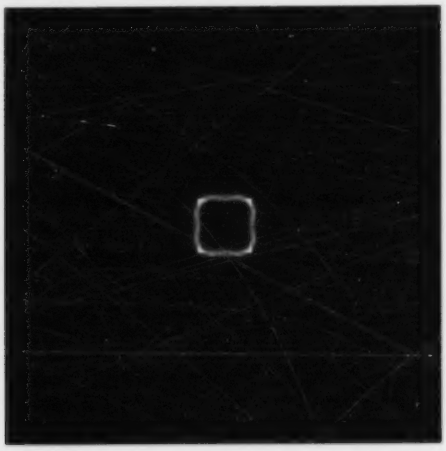

Figure 21. *The filtered result after applying a reverse FFT and removing the centring matrix.*

The simple example of a white cube on black background does not fully demonstrate the effect of a high-pass filter. Below is an example of a SAR image before and after the high-pass operation. Experimental observation shows that the image on the right will produce much more accurate results during the wake-detection process.

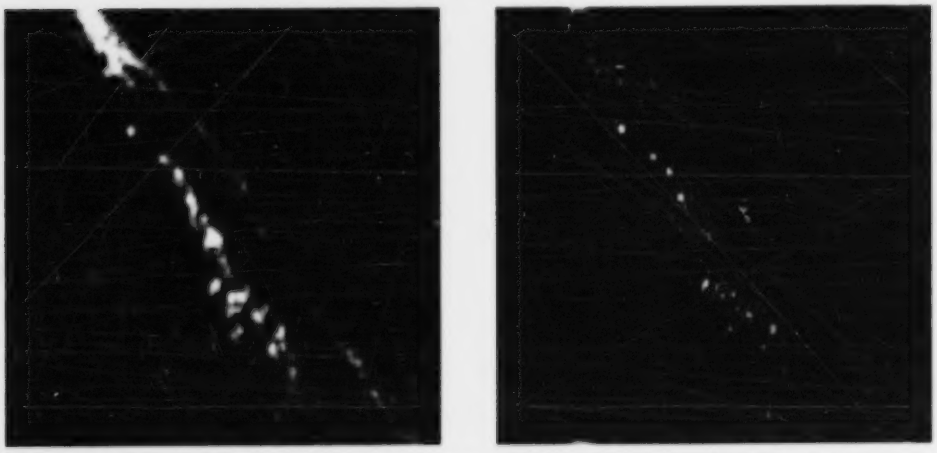

Figure 22. *SAR ship wake – Source image on left, High-pass filtered image on right.*

3.3.3 Wiener Filtering

The RedBox software makes use of a Wiener filtering algorithm to reduce the number of false alarms generated by the wake-detection algorithm. The Wiener filter is "an optimum nonrecursive linear estimator giving solutions to the Wiener-Hopf equation" [3] and can be used to amplify points in the Radon domain that have a high probability of being a wake arm while masking points that have a low probability of being a wake arm.

The Wiener filter algorithm is implemented in three parts. First, the Radon transform of an image is scanned for all bright and dark points with magnitude equal to or greater than a specified number of standard deviations from the mean. This point list will contain points representing the desired wake arms, but will also contain many points representing false alarms.

Once the list of points has been generated, the algorithm calculates the bright and dark filters, which are matrices representing the ideal shape of a radon peak generated by a strong wake arm feature. These bright and dark filters correspond to the magnitudes of a two-dimensional Gaussian peak centred on the matrix. The magnitude and width of the Gaussian peak are fixed at ideal values that were determined experimentally.

The final step of the Wiener filtering algorithm is to compare the grid region around each point in the list to the appropriate bright or dark filter (depending on the sign of the point). If there is a strong correlation between the shapes of the two matrices, the filter gives a high return and the magnitude of the point is increased proportionally. If there is little or no correlation between the filter and the point grid, the magnitude of the point is reduced proportionally.

4. Future Developments for Software

4.1 Ship Velocity Calculation

A foundation has been laid for incorporating the ability to calculate ship velocity and heading automatically from wake data. In order to calculate the ship velocity from the SAR ship wake it is necessary to know some key facts about the image, such as the radar grazing angle and the pixel spacing. Aside from this data, all other necessary information can be derived from the wake image. The software must be capable of measuring the separation between the ship centre and the apex of the wake, and must combine this measurement with geometrical data to determine the ship velocity. The current wake axis calculator is a primitive step toward that goal.

4.2 Refinement of Wake Detection Algorithm

The current wake detection algorithm is capable of successfully extracting significant linear features from an image. Development of a method to separate lines belonging to a wake from those based on unrelated features of the image will be necessary before user-independent analysis can become a reality. Since the RedBox software package is destined to become a fully automatic and user-independent wake analysis program, improvements to the algorithm will be required.

4.3 DLL Packaging

Many of the algorithms found in the RedBox tool may have useful applications in other simulation and analysis programs. It has been decided that the creation of a Windows dynamic link library (DLL) containing these algorithms would enable future developers to use this functionality in new programs. By pre-packaging efficient, error-free algorithms development time (and consequently, cost) of new software will be drastically reduced.

Another benefit of having a DLL of common algorithms arises when dealing with the concept of intellectual property. Since all source code for the algorithms will be compiled into one comprehensive package, any developers using the DLL will not have access to the code and therefore will not be able to 'borrow' or tweak the algorithms for their own use. This fact has great security advantages and could possibly open up new business opportunities for selling the DLL package without fear of competitors borrowing the algorithms for their own products.

4.4 Enhancing Interface Functionality and Aesthetics

Up until this point, the focus of development on the RedBox project has been to build a framework for automated wake analysis. Future versions of the software, while

continuing to improve the automation, should concentrate also on improving the functionality and aesthetics of the user interface. The current version of RedBox contains some artifacts from earlier development – buttons that no longer perform an action, fixed window sizes that could be dynamic, as well as other pitfalls for the first-time user. An experienced RedBox user will have learned which controls are useful and which are incomplete, however it will be necessary to tidy up the interface before the software can be released for everyday use.

Also, while most of the features of the RedBox application conform to Microsoft Windows standards, there are common features (such as copy and paste) that are currently not supported and would be very popular with users.

5. Observations

While developing the RedBox application, several important observations were made about computerized analysis of SAR images. For example, it was noted that not all radar images are equally well suited to being analyzed. Images containing coastlines and other significant background features often produce increased numbers of false alarms. Applying a high-pass filter often improves the result, however in many cases these features of the image cannot be totally removed. It is therefore extremely important to select a transform region that avoids these undesirable features.

Another observation was made concerning the differences in results after analysis of images from different satellites. Differences in the characteristics of the images cause over-detection or under-detection in cases where the filtering and detection algorithms are not optimized for the image type being analyzed. Images from satellites such as SEASAT usually require less filtering than those from RADARSAT or ERS. For this reason it was necessary to provide customizable filter parameters and detection bounds. The recommended SEASAT image filter has an outer radius between 5 and 10 pixels, while the suggested outer radius for RADARSAT or ERS image filtering is between 10 and 20 pixels.

6. Conclusions

Development of the RedBox application is still in progress. The current version of the RedBox software is mature enough to be a useful wake-analysis tool, but requires significant improvement in order to fulfill the target goal of being a fully automatic ship wake-processing tool.

Image processing algorithms such as the Fourier Transform and the Radon Transform allow the user to manipulate SAR images in a way that dramatically increases the chance of detecting ship wakes. A Wiener filtering algorithm has also been included, however the performance of the Wiener filter on SAR imagery has been inconsistent. Some images, after having the Wiener filter applied, show a marked improvement in wake arm detection while other images show little or no improvement after the same processing. Intelligent selection of the transform region was identified as an important step in the analysis procedure.

Future development of the RedBox application should focus on the aesthetic improvement of the software, as well as the development of a completely user-independent analysis. Manual analysis techniques should be retained for users who desire more flexibility than the automated process can provide.

7. References

1. Kain, Eugene. The MFC Answer Book, Addison Wesley Longman Inc, 1999.

2. Toft, Peter. The Radon Transform – Theory and Implementation, Ph.D. Thesis, Department of Mathematical Modelling, Technical University of Denmark, June 1996. 326 pages.

3. Tunaley, J.K.E., Jahans, P.A. and Dixon, J.A. "Evaluation of Radon Transform Techniques in the Detection and Enhancement of Linear Wake Features in Synthetic Aperture Radar (SAR) Wake Data – Part II", London Research and Development Corporation. Final Report under Contract File No. 003SV.W7714-8-5562/01-SV, March 1989.

Annex A – Screenshots

The following are some images captured from the screen while running the RedBox application. For information on the individual features of the application, refer to section 3.2 – Features.

Figure 23. *Opening a SAR bitmap image for processing*

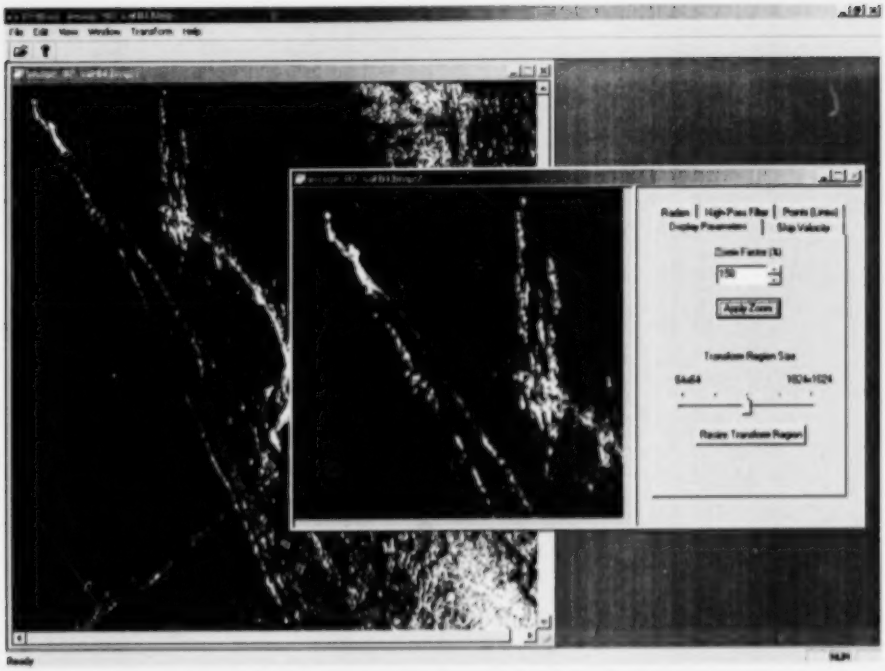

Figure 24. *The Transform view with the transform area on the original bitmap surrounded by a green box.*

Figure 25. *Performing a Radon transform – a progress bar appears at the bottom of the window pane.*

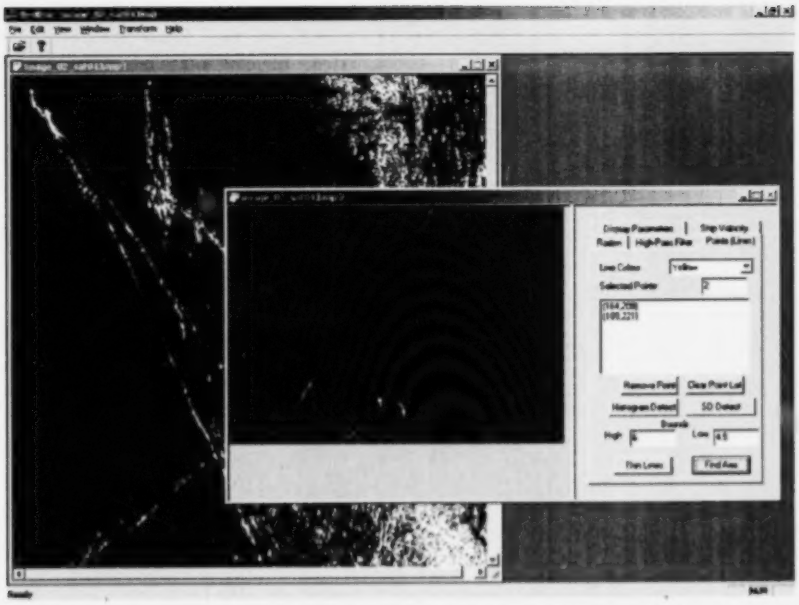

Figure 26. *The Radon transform complete – two points are selected, one on each wake arm. Notice the associated lines drawn in the transform area of the original image. The wake axis has also been calculated and is represented by a white line on the image.*

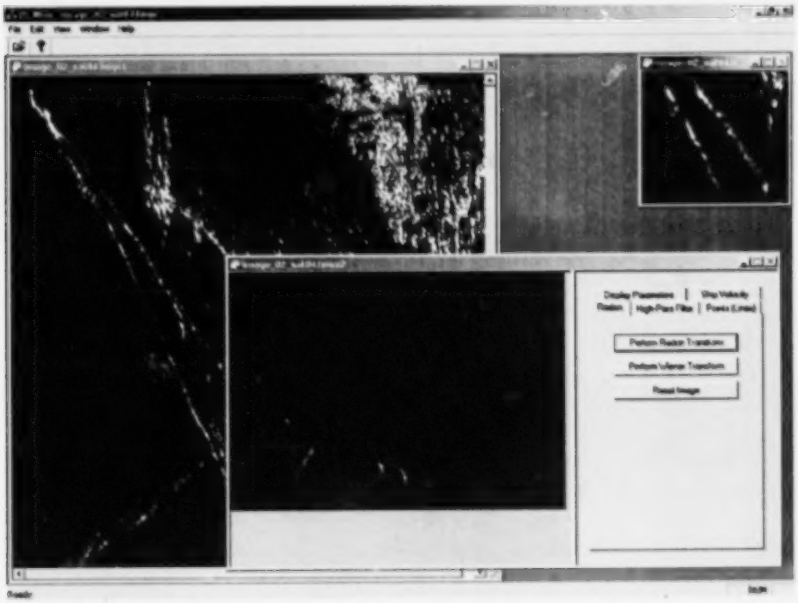

Figure 27. *Both the transform and zoom views are shown. It is possible to work with both views at once. The zoom view shows a magnified image of the area bounded by the red box.*

This page intentionally left blank.

Annex B – Microsoft Windows 24-Bit Bitmap file format

The Microsoft Windows 24-Bit Bitmap file format is composed of three distinct sections – a header describing the file attributes, a header describing the data attributes and the bit-mapped data itself.

File Header

(Excerpted from the Microsoft Developer Network Library, Visual Studio 6.0 release)

Visual C++ 6.0 defines a data structure that is used to store the file header information. This 14- byte structure is called BITMAPFILEHEADER and has the following format:

Format

```
typedef struct tagBITMAPFILEHEADER {
      WORD      bfType;
      DWORD     bfSize;
      WORD      bfReserved1;
      WORD      bfReserved2;
      DWORD     bfOffBits;
}BITMAPFILEHEADER;
```

Members

bfType
Specifies the file type. It must be BM.

bfSize
Specifies the size, in bytes, of the bitmap file.

bfReserved1
Reserved; must be zero.

bfReserved2
Reserved; must be zero.

bfOffBits
Specifies the offset, in bytes, from the BITMAPFILEHEADER structure to the bitmap bits.

Data Header

(Excerpted from the Microsoft Developer Network Library, Visual Studio 6.0 release)

Visual C++ 6.0 also defines a data structure that is used to store the image data attributes. The Microsoft help files can be quite misleading on this topic, as they often refer to the BITMAPCOREINFO structure which has a different purpose. The correct structure is called BITMAPINFOHEADER and has the following format:

Format

```
typedef struct tagBITMAPINFOHEADER {
      DWORD       biSize;
      LONG        biWidth;
      LONG        biHeight;
      WORD        biPlanes;
      WORD        biBitCount;
      DWORD       biCompression;
      DWORD       biSizeImage;
      LONG        biXPelsPerMeter;
      LONG        biYPelsPerMeter;
      DWORD       biClrUsed;
      DWORD       biClrImportant;
}BITMAPINFOHEADER;
```

Members

biSize:
Specifies the number of bytes required by the structure.

biWidth:
Specifies the width of the bitmap, in pixels.

biHeight:
Specifies the height of the bitmap, in pixels. If biHeight is positive, the bitmap is a bottom-up DIB and its origin is the lower-left corner. If biHeight is negative, the bitmap is a top-down DIB and its origin is the upper-left corner.

biPlanes:
Specifies the number of planes for the target device. This value must be set to 1.

biBitCount:
Specifies the number of bits per pixel. For 24-bit bitmaps, this value will always be 24. This indicates that the bitmap has a maximum of 2^24 colours. Each 3-byte triplet in the bitmap array represents the relative intensities of blue, green and red respectively, for a pixel.

biCompression:

Specifies the type of compression used on the image. Currently, RedBox does not offer full support for image compression.

biSizeImage:

Specifies the size, in bytes, of the image. This may be set to zero for uncompressed bitmaps.

biXPelsPerMeter:

Specifies the horizontal resolution, in pixels per meter, of the target device for the bitmap.

biYPelsPerMeter:

Specifies the vertical resolution, in pixels per meter, of the target device for the bitmap.

biClrUsed:

Specifies the number of colour indexes in the colour table that are actually used by the bitmap. If this value is zero, the bitmap uses the maximum number of colours corresponding to the value of the biBitCount member for the compression mode specified by biCompression.

biClrImportant:

Specifies the number of colour indexes that are required for displaying the bitmap. If this value is zero, all colours are required.

Data

The bit-mapped data in a 24-bit DIB is stored as an array of bytes, where each byte represents one of three colour intensities for a pixel. Each 3-byte group holds the values for the blue, green and red colour intensities for a pixel. Every scan line (horizontal line on the image) is DWORD-aligned, i.e. the number of bytes in each line must be padded so that it divides evenly by 4. It is not necessary to reverse the order of the scan lines since the MFC CBitmap class performs this conversion automatically when painted.

Annex C – Application Code

_CMPLX data structure

```
struct _cmplx
{
  float x,y;  //real and imaginary parts
};
```

Bitmap Format Conversion

```
bool CRedBoxDoc::ConvertToDIB(CString sFileName)
{

  DWORD width,height,bytewidth,padwidth;
  BYTE* pix;
  FILE* fin;
  FILE* fout;

  BITMAPFILEHEADER fileHeader;
  BITMAPINFOHEADER coreInfo;
  /*
  ------------------------------------------------------
    Read information from data file
  ------------------------------------------------------
  */

  //open the file that we want to convert
  fin = fopen(sFileName,"rb");
  //read the header information into bheader
  fread(&fileHeader,sizeof(BITMAPFILEHEADER),1,fin);
  //read the data attributes
  fread(&coreInfo,sizeof(BITMAPINFOHEADER),1,fin);
  //move the file pointer to the beginning of the bitmap data
  fseek(fin,fileHeader.bfOffBits,SEEK_SET);


  /*
  ------------------------------------------------------
    Calculations and Formatting
  ------------------------------------------------------

    Calclate the size of the bitmap array from width and
    height information. Remember: all scan lines on a bitmap
    are dword-aligned... multiples of 4 bytes.

    Add padding if necessary
  */

  width = coreInfo.biWidth;
  height = coreInfo.biHeight;
```

```
  padwidth = (3*width)%4;
  if(padwidth!=0)
    padwidth = 4-padwidth;
  bytewidth = 3*width+padwidth;

  /** Declare the bitmap array and read the
    image block from the file **/
  pix=new BYTE[bytewidth*height];
  fread(pix,bytewidth*height,1,fin);
  fclose(fin);



  fileHeader.bfSize=fileHeader.bfOffBits+bytewidth*height;
  coreInfo.biPlanes = 1;
  coreInfo.biSizeImage = bytewidth*height;
  coreInfo.biXPelsPerMeter = coreInfo.biYPelsPerMeter = 3780;

  /*
  ------------------------------------------------------
    Write formatted data back to the file
  ------------------------------------------------------
  */

  //open the same file for writing
  fout=fopen(sFileName,"wb");
  //write the file header
  fwrite(&fileHeader,sizeof(BITMAPFILEHEADER),1,fout);
  //write the info header
  fwrite(&coreInfo,sizeof(BITMAPINFOHEADER),1,fout);
  //write the block of bitmap data
  fwrite(pix,bytewidth*height,1,fout);
  //close the file - we're done
  fclose(fout);

  /** Don't Forget to free the memory from the data array.
    That would be a nasty leak! **/
  if(pix != NULL)
  {
    delete[] pix;
    pix=NULL;
  }
  return true;
}
```

Fourier Transform

```
void CRedBoxDoc::DoFFT(int isign)
{

   BITMAP bm;
   m_TransformBitmap.GetBitmap(&bm);

   if (bm.bmWidth != bm.bmHeight)
   {
      //cannot perform FFT on non-square region.
      return;
   }

   if(isign>0)
   {
      //forward FFT
      //centre the fourier spectrum using (1/-1) matrix
      SwapQuadrants();
   }

   FFT2D(m_cComplexThirdByteArray, bm.bmWidth, isign);

   if(isign<0)
   {
      //reverse FFT
      //remove the centering matrix
      SwapQuadrants();
   }

}

//=============================================================================
============================

void CRedBoxDoc::FFT2D(_cmplx *z, int nn, int isign)
{
   _cmplx *ampl;
   ampl=new _cmplx[nn];
   int  i,j;
   for(i=0;i<nn;i++)
   {
      for(j=0;j<nn;j++)
      {
         ampl[j].x=z[nn*i+j].x;
         ampl[j].y=z[nn*i+j].y;
      }
      FFT(ampl,nn,isign);
      for(j=0;j<nn;j++)
      {
         z[nn*i+j].x=ampl[j].x;
         z[nn*i+j].y=ampl[j].y;
```

```
    }
  } //end of i loop
//Start FFT in orthogonal direction
  for(j=0;j<nn;j++)
  {
    for(i=0;i<nn;i++)
    {
      ampl[i].x=z[nn*i+j].x;
      ampl[i].y=z[nn*i+j].y;
    }
    FFT(ampl,nn,isign);
    for(i=0;i<nn;i++)
    {
      z[nn*i+j].x=ampl[i].x;
      z[nn*i+j].y=ampl[i].y;
    }
  } //end of j loop
  delete[] ampl;
  return;
}

//=============================================================================
============================

void CRedBoxDoc::FFT(_cmplx *f, int count, int isign)
{
  float uu;
  _cmplx u,w,t;
  int i,j,k,kk,l,le,lel,n,nv,nm,ip,nn;

  // length of data
  n=count;
  // log (base 2) of length
  nn=(int)floor(log(n)/log(2.0)+1.0e-5);
  nv=n/2;
  nm=n-1;
  j=0;
  for (i=0;i<nm;i++)
  {
    if (i<j)
    {
      t.x=f[j].x;
      t.y=f[j].y;
      f[j].x=f[i].x;
      f[j].y=f[i].y;
      f[i].x=t.x;
      f[i].y=t.y;
    }
    k=nv;
    if (k<=j)
    {
      do
      {
        j=j-k;
```

```
        k=k/2;
      }
      while (k<=j);
    }
    j=j+k;
  }
  for (l=0;l<nn;l++)
  {
    le=1;
    for (kk=0;kk<l+1;kk++)
      le=2*le;
    le1=le/2;
    u.x=1.0f;
    u.y=0.0f;
    w.x=(float)cos(PI/le1);
    w.y=-isign*(float)sin(PI/le1);
    for (j=0;j<le1;j++)
    {
      i=j;
      if (i<n)
      {
        do
        {
          ip=i+le1;
          t.x=f[ip].x*u.x-f[ip].y*u.y;
          t.y=f[ip].x*u.y+f[ip].y*u.x;
          f[ip].x=f[i].x-t.x;
          f[ip].y=f[i].y-t.y;
          f[i].x=f[i].x+t.x;
          f[i].y=f[i].y+t.y;
          i=i+le;
        }
        while (i<n);
      }
      uu=u.x*w.x-u.y*w.y;
      u.y=u.y*w.x+u.x*w.y;
      u.x=uu;
    }
  }
  if (isign==-1)
    for (i=0;i<n;i++)
    {
      f[i].x=f[i].x/(float)n;
      f[i].y=f[i].y/(float)n;
    }
}
```

High-Pass Filter

```
void CRedBoxDoc::HighPassFilter(int nUpperBound, int nLowerBound)
{
  BITMAP bm;
  m_TransformBitmap.GetBitmap(&bm);
  int filterUpperBound, filterLowerBound;

  int i,j;
  double r, filter;


  //do filter stuff

  if (nUpperBound<=nLowerBound)
  {
    TRACE0("HighPassFilter - Upper Bound below Lower Bound");
  }

  //convert bounds into image size percentages - so that
  //different sized images don't filter differently

  filterUpperBound = (int)(bm.bmWidth * (nUpperBound/100.0));
  filterLowerBound = (int)(bm.bmWidth * (nLowerBound/100.0));

  int N = bm.bmWidth;
  for (i = 0; i<N; i++)
  {
    for (j = 0; j < N; j++)
    {
      r = sqrt((i-(N/2))*(i-(N/2)) + (j-(N/2))*(j-(N/2)));
      if (r > filterUpperBound)
      {
        filter = 1.0;
      }
      else if (r<filterLowerBound)
      {
        filter = 0;
      }
      else
      {
        filter = (1.0 - cos((PI*(r -
filterLowerBound))/(float)(filterUpperBound-filterLowerBound))) /
2.0;
      }
      m_cComplexThirdByteArray[N*i + j].x *= (float)filter;
      m_cComplexThirdByteArray[N*i + j].y *= (float)filter;
    }
  }


}
```

Radon Transform

```
void CRedBoxDoc::DoRadon(CProgressCtrl *pProgressBar)
{
   //don't transform if already transformed
   if (GetShowingRadon()) return;

   //set radon flag so we don't transform again.
   SetShowingRadon(true);


   /*
   IMPORTANT!:  At this stage of development,
   the radon algorithm MUST be given a SQUARE
   transform region, otherwise it will explode.
   */

   BITMAP bm;
   m_TransformBitmap.GetBitmap(&bm);

   //initialize progress bar limits and stepsize
   pProgressBar->SetRange(0,(short)bm.bmWidth);
   pProgressBar->SetStep(1);

   //calculate resulting bitmap dimensions
   //newWidth (=rho) will equal distance between
   //opposite corners on original image
   int newWidth = (int)(bm.bmWidth*sqrt(2));
   int newHeight = bm.bmHeight;


   //declare an array for holding results temporarily
   _cmplx* temp;
   temp = new _cmplx[newWidth*newHeight];


   //RADON TRANSFORM BEGINS
   //----------------------


   int i, j, k;        //counters
   int numPixels;      //number of pixels in the original image
   double xmax, ymax;
   bool isVertical, isHorizontal, validEndpoint1, validEndpoint2;
   double result;
   double rightXparam, leftXparam, topYparam, bottomYparam;
   double rhoMax;      //maximum displacement from image centre
   double currentRho;    //current displacement from image centre
   double deeTheta;    //angular increment per pixel
   double currentTheta; //the angle currently being evaluated
   double sum,avg;
   CPoint p1, p2;
   double cursorX, cursorY;
```

```
//calculate average value of bitmap
numPixels = bm.bmHeight*bm.bmWidth;

sum = 0;
for (i = 0; i < numPixels; i++)
{
   sum+=m_cComplexThirdByteArray[i].x;
}
avg = sum/(double)numPixels;


//find extents of image, from centre
xmax = (bm.bmWidth - 1)/2;
ymax = (bm.bmHeight - 1)/2;


//find largest displacement from centre
//(Radial distance from centre to a corner)
rhoMax = ((bm.bmWidth*sqrt(2)) -1 ) / 2;


//find angular increment
deeTheta = PI/newHeight;

for (i = 0; i<newHeight;i++)
{
   isVertical = false;
   isHorizontal = false;

   //calculate angle currently being used
   currentTheta = i * deeTheta;

   //if sin or cosine is very small, pretend it is zero.
   if (fabs(sin(currentTheta))<0.00001)
   {
      isVertical = true;
   }
   else if (fabs(cos(currentTheta))<0.00001)
   {
      isHorizontal = true;
   }

   //start at extreme negative rho
   currentRho = -(newWidth - floor(rhoMax));
   for (j = 0; j< newWidth; j++)
   {
      if((fabs(currentRho*cos(currentTheta)) <= xmax ) &&
(fabs(currentRho*sin(currentTheta)) <= ymax))
      {
         /*
         Find endpoints on image, using parametric
         equations as follows:
```

```
            X=SIN(THETA)*PARAMETER+RHO*COS(THETA)
            Y=-COS(THETA)*PARAMETER+RHO*SIN(THETA)
          */
          if (!isVertical)
          {
            rightXparam = ((xmax-
currentRho*cos(currentTheta))/sin(currentTheta));
            leftXparam = ((-xmax -
currentRho*cos(currentTheta))/sin(currentTheta));
          }
          if (!isHorizontal)
          {
            topYparam = ((currentRho*sin(currentTheta) -
ymax)/cos(currentTheta));
            bottomYparam = ((currentRho*sin(currentTheta) +
ymax)/cos(currentTheta));
          }

          /*
          initialize booleans - we don't have any
          valid endpoints yet
          */
          validEndpoint1 = validEndpoint2 = false;

          /*
          If not vertical, check to see if left
          and right edge points are also within
          top and bottom bounds.  If so, they're
          valid.
          */

          double yRight, yLeft;
          double point1Param, point2Param;
          if (!isVertical)
          {
            yRight = -
cos(currentTheta)*rightXparam+currentRho*sin(currentTheta);
            yLeft = -
cos(currentTheta)*leftXparam+currentRho*sin(currentTheta);
            if (fabs(yRight) <= ymax)
            {
              point1Param= rightXparam;
              validEndpoint1 = true;
            }
            if (fabs(yLeft) <= ymax)
            {
              if (!validEndpoint1)
              {
                point1Param = leftXparam;
                validEndpoint1 = true;
              }
              else
              {
                point2Param = leftXparam;
```

```
                validEndpoint2 = true;
              }
            }
          }


          double xTop, xBottom;
          if (!isHorizontal)
          {
            xTop =
sin(currentTheta)*topYparam+currentRho*cos(currentTheta);
            xBottom =
sin(currentTheta)*bottomYparam+currentRho*cos(currentTheta);
            if (fabs(xTop) <= xmax)
            {
              if (!validEndpoint2)
              {
                if (!validEndpoint1)
                {
                  point1Param = topYparam;
                  validEndpoint1 = true;
                }
                else
                {
                  point2Param = topYparam;
                  validEndpoint2 = true;
                }
              }
            }
            if (fabs(xBottom) <= xmax)
            {
              if (!validEndpoint2)
              {
                if (!validEndpoint1)
                {
                  point1Param = bottomYparam;
                  validEndpoint1 = true;
                }
                else
                {
                  point2Param = bottomYparam;
                  validEndpoint2 = true;
                }
              }
            }
          }

          /*
          Check to see if we have two valid endpoints.
          At this point, we always should
          */
          if(validEndpoint1&&validEndpoint2)
          {
```

```
            p1.x =
(long)(sin(currentTheta)*point1Param+(currentRho*cos(currentTheta))+x
max);
            p2.x =
(long)(sin(currentTheta)*point2Param+(currentRho*cos(currentTheta))+x
max);
            p1.y = (long)(-
cos(currentTheta)*point1Param+(currentRho*sin(currentTheta))+ymax);
            p2.y = (long)(-
cos(currentTheta)*point2Param+(currentRho*sin(currentTheta))+ymax);


            //sum and average along this line...

            //find change along each axis
            double dx = p2.x - p1.x;
            double dy = p2.y - p1.y;

            //calculate the number of steps required
            int numSteps = (int)sqrt(dx*dx + dy*dy);
            int linePixels;
            double xStepSize;
            double yStepSize;


            if(numSteps>0)
            {
              xStepSize = dx/numSteps;
              yStepSize = dy/numSteps;
            }
            else
            {
              xStepSize = 0;
              yStepSize = 0;
            }


            //set cursor to start point
            cursorX = p1.x;
            cursorY = p1.y;

            //number of pixels is one greater than the number of
steps
            linePixels = numSteps +1;

            //initialize line value to zero before summing
            result=0;
            for (k=0; k<linePixels; k++)
            {
              result +=
m_cComplexThirdByteArray[(bm.bmWidth*(int)cursorY) + (int)cursorX].x;
              cursorX += xStepSize;
              cursorY += yStepSize;
            }
```

```
            //divide by square root of number of pixels to find
average value.
            //techically it's not an average, but it helps to prevent
the natural bias
            //toward short lines
            result/=(float)sqrt(linePixels);
          }
          else
          {
            //didn't find two valid endpoints, so just use average
value for now
            result = avg;
          }
        }
        else
        {
          //line is off the screen, so just use the image average
instead;
          result = avg;
        }
        //store it in the data array
        temp[(newWidth * i) + j].x = (float)result;

        //move to the next displacement
        currentRho++;


      }

      pProgressBar->StepIt();

    }


    //RADON TRANSFORM COMPLETE
    //------------------------
    //make a new, bigger complex array,
    //fill it, and regenerate the new bitmap

    if(m_cComplexThirdByteArray != NULL)
    {
      delete[] m_cComplexThirdByteArray;
      m_cComplexThirdByteArray = NULL;
    }
    m_cComplexThirdByteArray = new _cmplx[newWidth*newHeight];

    CopyMemory(m_cComplexThirdByteArray, temp,
newWidth*newHeight*sizeof(_cmplx));

    //generate a bitmap from this new data
    GenerateBitmapFromThirdBytes(newWidth, newHeight);

    //clean up unwanted memory
```

```
    if(temp != NULL)
    {
      delete [] temp;
      temp = NULL;
    }
}
```

Wiener Filter

```
void CRedBoxDoc::DoWiener(CProgressCtrl *pProgressBar)
{

  double** pyy;
  double** pinv;
  double* pxy;
  double* hp;
  double* hn;
  double* y;

  //size of the box
  const int dimension = 7;



  int half, hq, n, i, j, ii, jj, ind1, ind2, k, numPeaks, dx1, dy1,
dx2, dy2;
  int bw, dw;
  double c, newValue, sdp, sdn, rad1, rad2, snp, snn, gi, gj;

  BITMAP bm;
  m_TransformBitmap.GetBitmap(&bm);



  //initialize data structures
  pyy = new double* [dimension*dimension];
  pinv = new double* [dimension*dimension];
  for (i=0; i<dimension*dimension; i++)
  {
    pyy[i] = new double[dimension*dimension];
    pinv[i] = new double[dimension*dimension];
  }

  pxy = new double[dimension*dimension];
  hp = new double[dimension*dimension];
  hn = new double[dimension*dimension];
  y = new double[dimension*dimension];

  bw=25;
  dw=30;
  c=2;
  sdp=bw/4.0/c;
  sdn=dw/4.0/c;
  snp=5.0;
  snn=5.0;

  half=dimension/2;

  sdp=10.0;
```

```
  n= dimension;
  hq= half+1;

  CList<CRadonPoint,CRadonPoint> ValidPeakList;
  CList<CRadonPoint,CRadonPoint> WienerPeakList;

  //detect peaks that are wakes(>6sd), and those that may need
wiener filtering(>4.5sd)
  numPeaks = FindPeaks(6,&ValidPeakList,4.5, &WienerPeakList);
  pProgressBar->SetRange(0, WienerPeakList.GetCount());
  pProgressBar->SetStep(1);


  //Calculating Bright Filter
  for (i=0; i<n; i++)
  {
    for (j=0; j<n; j++)
    {
      for (ii=0; ii<n; ii++)
      {
        for (jj=0; jj<n; jj++)
        {
          ind1=(i)*n+j;
          ind2=(ii)*n+jj;
          dx1=i-hq;
          dy1=j-hq;
          dx2=ii-hq;
          dy2=jj-hq;
          rad1=sqrt(dx1*dx1+dy1*dy1);
          rad2=sqrt(dx2*dx2+dy2*dy2);
          gi=exp(-(rad1*rad1)/(2.0*sdp*sdp));
          gj=exp(-(rad2*rad2)/(2.0*sdp*sdp));
          pyy[ind1][ind2]=snp*snp*(gi*gj);
          if(ind1==(n*n) / 2 + 1)
          {
            pxy[ind2]=pyy[ind1][ind2];
          }
          if(ind1==ind2)
          {
            pyy[ind1][ind2]=pyy[ind1][ind2]+1.0;
          }
        }
      }
    }
  }

  InvertMatrix(pyy,pinv,n*n);


  for (i=0; i<n*n; i++)
  {
    hp[i]=0.0;
  }
```

```
for (j=0; j<n*n; j++)
{
  for (k=0; k<n*n; k++)
  {
    hp[j]=hp[j]+pxy[k]*pinv[k][j];
  }
}


//Calculating Dark Filter
for (i=0; i<n; i++)
{
  for (j=0; j<n; j++)
  {
    for (ii=0; ii<n; ii++)
    {
      for (jj=0; jj<n; jj++)
      {
        ind1=i*n+j;
        ind2=ii*n+jj;
        dx1=i-hq;
        dy1=j-hq;
        dx2=ii-hq;
        dy2=jj-hq;
        rad1=sqrt(dx1*dx1+dy1*dy1);
        rad2=sqrt(dx2*dx2+dy2*dy2);
        gi=exp(-(rad1*rad1)/(2.0*sdn*sdn));
        gj=exp(-(rad2*rad2)/(2.0*sdn*sdn));
        pyy[ind1][ind2]=snn*snn*(gi*gj);
        if(ind1==((n*n) / 2)+1)
        {
          pxy[ind2]=pyy[ind1][ind2];
        }
        if(ind1==ind2)
        {
          pyy[ind1][ind2]=pyy[ind1][ind2]+1.0;
        }
      }
    }
  }
}
InvertMatrix(pyy,pinv,n*n);

for (i=0; i<n*n; i++)
{
  hn[i]=0.0;
}
for (j=0; j<n*n; j++)
{
  for (k=0; k<n*n; k++)
  {
    hn[j]=hn[j]+pxy[k]*pinv[k][j];
  }
}
```

```
  CRadonPoint currentValue;
  POSITION peakIterator;
  peakIterator = WienerPeakList.GetHeadPosition();

  while(peakIterator != NULL)
  {
    currentValue = WienerPeakList.GetNext(peakIterator);
    for (i=0; i<n*n; i++)
    {
      y[i]=0.0;
    }

    int bitmapWidth = bm.bmWidth;
    int bitmapHeight = bm.bmHeight;
    int boxCentre;
    int topLeftCorrection;

    //check to see if the box is totally on the image horizontally
    if ((currentValue.x/GetTransformZoomFactor() > (dimension/2+1))
&& (currentValue.x/GetTransformZoomFactor() < (bitmapWidth -
(dimension/2+1))))
    {
      //check to see if the box is totally on the image vertically
      if ((currentValue.y/GetTransformZoomFactor() >
(dimension/2+1)) && (currentValue.y/GetTransformZoomFactor() <
(bitmapHeight - (dimension/2+1))))
      {
        boxCentre =
(int)(((currentValue.y)/GetTransformZoomFactor())*bitmapWidth) +
(int)((currentValue.x)/GetTransformZoomFactor());
        topLeftCorrection = ((dimension/2) + 1)*bitmapWidth +
(dimension/2)+1;
        for (i=0; i<dimension; i++)
        {
          for (j=0; j<dimension; j++)
          {
            y[i*dimension + j] = m_cComplexThirdByteArray[boxCentre
- topLeftCorrection + i*bitmapWidth + j].x;
          }
        }

        if(m_cComplexThirdByteArray[boxCentre].x>0.0)
        {
          newValue=0.0;
          for (i=0; i<n*n; i++)
          {
            newValue=newValue+hp[i]*y[i];
          }
        }
        else
        {
```

```
            newValue=0.0;
            for (i=0; i<n*n; i++)
            {
              newValue=newValue+hn[i]*y[i];
            }
          }
          m_cComplexThirdByteArray[boxCentre].x = (float)newValue;

        }
      }
      pProgressBar->StepIt();
    }

    GenerateBitmapFromThirdBytes(bm.bmWidth, bm.bmHeight);
    //free data structure memory

    for (i=0; i<dimension*dimension; i++)
    {
      delete[] pyy[i];
      delete[] pinv[i];
    }

    if (pyy != NULL)
    {
      delete[] pyy;
      pyy = NULL;
    }
    if (pinv != NULL)
    {
      delete[] pinv;
      pinv = NULL;
    }
    if (pxy != NULL)
    {
      delete[] pxy;
      pxy = NULL;
    }
    if (hp != NULL)
    {
      delete[] hp;
      hp = NULL;
    }
    if (hn != NULL)
    {
      delete[] hn;
      hn = NULL;
    }
    if (y != NULL)
    {
      delete[] y;
      y = NULL;
    }
}
```

```
//=============================================================================
============================

void CRedBoxDoc::InvertMatrix(double** b, double** a, int n)
{
  int i,j,k;
  double d;
  int* indx;
  double* vector;
  indx = new int[n*n];
  vector = new double[n*n];
  //set up identity matrix
  for (i=0; i<n; i++)
  {
    for (j=0; j<n; j++)
    {
      a[i][j] = 0.0;
    }
    a[i][i] = 1.0;
  }


  d = 0.0;
  ludcmp(b,n,indx,d);
  for(j=0;j<n; j++)
  {
    for (k=0; k<n; k++)
    {
      vector[k] = a[k][j];
    }
    lubksb(b,n,indx,vector);
    for (k=0; k<n; k++)
    {
      a[k][j] = vector[k];
    }
  }

  if (vector!=NULL)
  {
    delete[] vector;
    vector=NULL;
  }
  if (indx!=NULL)
  {
    delete[] indx;
    indx=NULL;
  }


}

//=============================================================================
============================
```

```
void CRedBoxDoc::ludcmp(double **a, int n, int *indx, double d)
{

  const double TINY = 1.0e-20;

  int i, j, k, imax;
  double aamax, sum, dum;

  double* vv;
  vv = new double[n*n];

  d = 1.0;
  for(i=0; i<n; i++)
  {
    aamax = 0.0;
    for (j=0; j<n; j++)
    {
      if (fabs(a[i][j])>aamax)
      {
        aamax = fabs(a[i][j]);
      }
    }
    if (aamax == 0.0)
    {
      TRACE0("Singular Matrix");
      return;
    }
    vv[i] = 1.0/aamax;    //scaling factor
  }

  for (j=0; j<n; j++)
  {
    if (j>0)
    {
      for (i=0; i<j; i++)
      {
        sum = a[i][j];
        if (i>0)
        {
          for (k=0; k<i; k++)
          {
            sum=sum-(a[i][k]*a[k][j]);
          }
          a[i][j] = sum;
        }
      }
    }
    aamax = 0.0;

    for (i=j; i<n; i++)
    {
      sum=a[i][j];
      if (j>0)
```

```
        {
          for (k=0; k<j; k++)
          {
            sum=sum-a[i][k]*a[k][j];
          }
          a[i][j] = sum;
        }
        dum=vv[i]*fabs(sum);
        if(dum>=aamax)
        {
          imax=i;
          aamax=dum;
        }
      }
      if (j!=imax)
      {
        for (k=0; k<n; k++)
        {
          dum=a[imax][k];
          a[imax][k]=a[j][k];
          a[j][k]=dum;
        }
        d=(-d);
        vv[imax]=vv[j];
      }
      indx[j]=imax;
      if(j!=(n-1))
      {
        if(a[j][j]==0.0)
        {
          a[j][j]=TINY;
        }
        dum=1.0/a[j][j];
        for(i=j+1;i<n; i++)
        {
          a[i][j] = a[i][j]*dum;
        }
      }
    }
    if(a[n-1][n-1]==0.0)
    {
      a[n-1][n-1] = TINY;
    }

    if (vv!=NULL)
    {
      delete[] vv;
      vv=NULL;
    }
}

//=============================================================================
============================
```

```
void CRedBoxDoc::lubksb(double **a, int n, int *indx, double *b)
{
  int i, j, ii, ll;
  double sum;

  ii=-1;

  for (i=0; i<n; i++)
  {
    ll=indx[i];
    sum=b[ll];
    b[ll]=b[i];      //MISSING LINE!
    if (ii!=-1)
    {
      for(j=ii;j<=(i-1);j++)
      {
        sum=sum-a[i][j]*b[j];
      }
    }
    else
    {
      if(sum!=0)
      {
        ii=i;
      }
    }
    b[i]=sum;
  }
  for(i=(n-1);i>=0;i--)
  {
    sum=b[i];
    if(i<(n-1))
    {
      for(j=(i+1);j<n;j++)
      {
        sum=sum-a[i][j]*b[j];
      }
    }
    b[i]=sum/a[i][i];
  }
}
```

Line Detection (histogram)

```
void CRedBoxDoc::DoPixValueHistogram(double cutoff1,
CList<CRadonPoint,CRadonPoint> *bin1, double cutoff2,
CList<CRadonPoint,CRadonPoint> *bin2, COLORREF lineColour)
{
  double upperValue, lowerValue;
  GetPixelCutoffBounds(cutoff1,lowerValue, cutoff2, upperValue);
  int i;

  BITMAP bm;
  m_TransformBitmap.GetBitmap(&bm);


  for (i = 0; i<(bm.bmWidth*bm.bmHeight);i++)
  {
    if(m_cComplexThirdByteArray[i].x<lowerValue)
    {
      CRadonPoint temp((i%bm.bmWidth),(i/bm.bmWidth), lineColour);
      double rhoMax = (bm.bmHeight*sqrt(2))/2;
      double rho = temp.x;
      temp.SetRho(rho-rhoMax);
      double theta = PI*((temp.y)/(double)bm.bmHeight);
      temp.SetTheta(theta);
      bin1->AddTail(temp);

    }
    if(m_cComplexThirdByteArray[i].x>upperValue)
    {
      CRadonPoint temp((i%bm.bmWidth),(i/bm.bmWidth), lineColour);
      double rhoMax = (bm.bmHeight*sqrt(2))/2;
      double rho = temp.x;
      temp.SetRho(rho-rhoMax);
      double theta = PI*((temp.y)/(double)bm.bmHeight);
      temp.SetTheta(theta);
      bin2->AddTail(temp);
    }
  }
}

//=================================================================
============================

void CRedBoxDoc::GetPixelCutoffBounds(double lowCutoff, double&
lowValue,double highCutoff, double& highValue)
{
  BITMAP bm;
  m_TransformBitmap.GetBitmap(&bm);

  int numValues = bm.bmWidth*bm.bmHeight;
  _cmplx* complexValueArray = new _cmplx[numValues];
```

```
  //prepare and copy data to temporary sorting array
  ZeroMemory(complexValueArray, sizeof(_cmplx)*numValues);
  CopyMemory(complexValueArray, m_cComplexThirdByteArray,
sizeof(_cmplx)* numValues);

  /* Sort pixels using Quicksort algorithm: */
  qsort((void *) complexValueArray, (size_t)numValues,
sizeof(_cmplx), (int(*)(const void*, const
void*))CRedBoxDoc::CompareComplexReal);

  //find pixel values at cutoff boundaries
  int lowIndex = (int)floor(numValues*lowCutoff);
  int highIndex = (int)floor(numValues*highCutoff);

  //return them
  lowValue = complexValueArray[lowIndex].x;
  highValue = complexValueArray[highIndex].x;

  //free memory
  if (complexValueArray != NULL)
  {
    delete [] complexValueArray;
    complexValueArray = NULL;
  }

}

//=================================================================
============================

int CRedBoxDoc::CompareComplexReal(const void *arg1, const void
*arg2)
{
  //compare the real part of the complex numbers
  //used in quicksort algorithm
  return (int)((*(_cmplx*)arg1).x - (*(_cmplx*)arg2).x);
}
```

Line Detection (standard deviation)

```
int CRedBoxDoc::FindPeaks(double cutoff1, CList<CRadonPoint,
CRadonPoint> *ValidPeakList, double cutoff2,
CList<CRadonPoint,CRadonPoint> *WienerPeakList, COLORREF lineColour1,
COLORREF lineColour2 )
{
  int i, numPixels, unusedPixels;
  double imageMean, imageVariance, imageSum, imageSquareSum;
  double sd, highCutoff, lowCutoff;
  BITMAP bm;
  m_TransformBitmap.GetBitmap(&bm);

  numPixels = bm.bmWidth*bm.bmHeight;
  unusedPixels = 0;


  //find image mean
  imageMean = 0;
  imageVariance = 0;
  imageSum = 0;
  imageSquareSum = 0;

  int leftBoundary, rightBoundary;

  leftBoundary = (int)((bm.bmWidth/2) - ((bm.bmWidth/sqrt(2))/2) +
1);
  rightBoundary = (int)((bm.bmWidth/2) + ((bm.bmWidth/sqrt(2))/2) +
1);

  for (i =0; i<numPixels; i++)
  {
    if(((i%bm.bmWidth) > leftBoundary) && ((i%bm.bmWidth) <=
rightBoundary))
    {
      imageSum += m_cComplexThirdByteArray[i].x;
      imageSquareSum += (float)pow(m_cComplexThirdByteArray[i].x,2);
    }
    else
    {
      //the pixel falls in the 'scalloped' region of the radon
transform - ignore it.
      unusedPixels++;
    }
  }
  imageMean = imageSum/(numPixels - unusedPixels);
  imageVariance = (imageSquareSum/(numPixels - unusedPixels)) -
imageMean*imageMean;
  sd = sqrt(imageVariance);


  highCutoff = cutoff1*sd;
```

```
  lowCutoff = cutoff2*sd;

  for (i=0; i<numPixels; i++)
  {
    if (fabs(m_cComplexThirdByteArray[i].x) > highCutoff)
    {

      CRadonPoint temp((i%bm.bmWidth),(i/bm.bmWidth), lineColour1);
      double rhoMax = (bm.bmHeight*sqrt(2))/2;
      double rho = temp.x;
      temp.SetRho(rho-rhoMax);
      double theta = PI*((temp.y)/(double)bm.bmHeight);
      temp.SetTheta(theta);
      ValidPeakList->AddTail(temp);
    }
    else if (fabs(m_cComplexThirdByteArray[i].x) > lowCutoff)
    {
      CRadonPoint temp((i%bm.bmWidth),(i/bm.bmWidth), lineColour2);
      double rhoMax = (bm.bmHeight*sqrt(2))/2;
      double rho = temp.x;
      temp.SetRho(rho-rhoMax);
      double theta = PI*((temp.y)/(double)bm.bmHeight);
      temp.SetTheta(theta);
      WienerPeakList->AddTail(temp);
    }
  }


  return (ValidPeakList->GetCount() + WienerPeakList->GetCount());
}
```

Drawing Radon lines

```
void CRedBoxDoc::DrawRadonLine(CDC* pDC, double theta, double rho,
int targetDimension, float factor, int offsetLeft, int offsetTop,
CPen * thePen)
{

   ASSERT_VALID(pDC);
   ASSERT_VALID(thePen);
   ASSERT(factor > 0);
   ASSERT(theta <=PI);

   CPoint p1, p2;

   pDC->SelectObject(thePen);


   bool isVertical, isHorizontal, validEndpoint1, validEndpoint2;
   double rightXparam, leftXparam, topYparam, bottomYparam;
   double xmax, ymax;

   xmax = (targetDimension - 1)/2;
   ymax = (targetDimension - 1)/2;

   isVertical = false;
   isHorizontal = false;

   if (fabs(sin(theta))<0.00001)
   {
      isVertical = true;
   }
   else if (fabs(cos(theta))<0.00001)
   {
      isHorizontal = true;
   }
   /*
   Find endpoints on image, using parametric equations as follows:
      X=SIN(THETA)*PARAMETER+RHO*COS(THETA)
      Y=-COS(THETA)*PARAMETER+RHO*SIN(THETA)
   */

   if (!isVertical)
   {
      rightXparam = ((xmax-rho*cos(theta))/sin(theta));
      leftXparam = ((-xmax - rho*cos(theta))/sin(theta));
   }
   if (!isHorizontal)
   {
      topYparam = ((rho*sin(theta) - ymax)/cos(theta));
      bottomYparam = ((rho*sin(theta) + ymax)/cos(theta));
   }
```

```
//initialize booleans - we don't have any valid endpoints yet
validEndpoint1 = validEndpoint2 = false;

/*
If not vertical, check to see if left and right edge points
are also within top and bottom bounds.  If so, they're valid.
*/

double yRight, yLeft;
double point1Param, point2Param;
if (!isVertical)
{
  yRight = -cos(theta)*rightXparam+rho*sin(theta);
  yLeft = -cos(theta)*leftXparam+rho*sin(theta);
  if (fabs(yRight) <= ymax)
  {
    point1Param= rightXparam;
    validEndpoint1 = true;
  }
  if (fabs(yLeft) <= ymax)
  {
    if (!validEndpoint1)
    {
      point1Param = leftXparam;
      validEndpoint1 = true;
    }
    else
    {
      point2Param = leftXparam;
      validEndpoint2 = true;
    }
  }
}

double xTop, xBottom;
if (!isHorizontal)
{
  xTop=sin(theta)*topYparam+rho*cos(theta);
  xBottom=sin(theta)*bottomYparam+rho*cos(theta);
  if (fabs(xTop) <= xmax)
  {
    if (!validEndpoint2)
    {
      if (!validEndpoint1)
      {
        point1Param = topYparam;
        validEndpoint1 = true;
      }
      else
      {
        point2Param = topYparam;
        validEndpoint2 = true;
      }
    }
```

```
      }
      if (fabs(xBottom) <= xmax)
      {
        if (!validEndpoint2)
        {
          if (!validEndpoint1)
          {
            point1Param = bottomYparam;
            validEndpoint1 = true;
          }
          else
          {
            point2Param = bottomYparam;
            validEndpoint2 = true;
          }
        }
      }
    }

    if(validEndpoint1&&validEndpoint2)
    {
      p1.x=(long)(sin(theta)*point1Param+(rho*cos(theta))+xmax);
      p2.x=(long)(sin(theta)*point2Param+(rho*cos(theta))+xmax);
      p1.y=(long)(-cos(theta)*point1Param+(rho*sin(theta))+ymax);
      p2.y=(long)(-cos(theta)*point2Param+(rho*sin(theta))+ymax);
    }


    p1.x += offsetLeft;
    p2.x += offsetLeft;
    p1.y += offsetTop;
    p2.y += offsetTop;

    pDC->MoveTo(p1);
    pDC->LineTo(p2);


}
```

Generating the Complex Data Array from a Bitmap

```
void CRedBoxDoc::GenerateThirdByteArray()
{
  //synchronize the third byte array with the bitmap that is
currently in memory

  //clear and populate ThirdByteArray here

  BITMAP bm;
  m_TransformBitmap.GetBitmap(&bm);
  int numBytes = bm.bmWidthBytes * bm.bmHeight;


  if(m_cComplexThirdByteArray != NULL)
  {
    delete[] m_cComplexThirdByteArray;
    m_cComplexThirdByteArray=NULL;
  }

  m_cComplexThirdByteArray = new _cmplx[numBytes/ 3];

  BYTE* fullByteArray = new BYTE[numBytes];
  m_TransformBitmap.GetBitmapBits(numBytes, fullByteArray);

  for (int i = 0;i < numBytes; i++)
  {
    if (i % 3 == 2)    //grab the third byte of each pixel
    {
      m_cComplexThirdByteArray[i / 3].x = fullByteArray[i];
      m_cComplexThirdByteArray[i / 3].y = fullByteArray[i];
    }
  }

  if(fullByteArray != NULL)
  {
    delete[] fullByteArray;
    fullByteArray=NULL;
  }
}
```

Generating a Bitmap from the Complex Data Array (real part only)

```
void CRedBoxDoc::GenerateBitmapFromThirdBytes(int bitmapWidth, int
bitmapHeight)
{
   int i;
   float factor, max, min;

   int padwidth = (3*bitmapWidth)%4;
   if(padwidth!=0)
     padwidth = 4-padwidth;
   int widthBytes = 3*bitmapWidth+padwidth;

   BYTE* fullByteArray;
   fullByteArray = new BYTE[widthBytes * bitmapHeight];


   //find max and min points in the bit array
   max = 0;
   min = 255;

   for (i=0; i<bitmapWidth*bitmapHeight; i++)
   {
     if(m_cComplexThirdByteArray[i].x > max)
     {
       max = m_cComplexThirdByteArray[i].x;
     }
     else if (m_cComplexThirdByteArray[i].x < min)
     {
       min = m_cComplexThirdByteArray[i].x;
     }
   }
   //determine scaling factor
   factor = 255/(float)(max - min);


   //create full bitmap bit array
   int padding, padAmount;
   padding = 0;
   padAmount =  widthBytes - (bitmapWidth * 3);

   int j;

   for (i = 0; i < bitmapHeight; i++)
   {
     for (j=0; j<(bitmapWidth*3); j++)
     {
       fullByteArray[(i*widthBytes)+j] =
(BYTE)((m_cComplexThirdByteArray[(((bitmapHeight-
(i+1))*bitmapWidth*3) + j)/ 3].x - min) * factor) ;
```

```
    }
  }


  //create and populate a bitmap structure and handle
  HBITMAP     hbm;
  BITMAPINFO  bmi;
  BYTE* pclrBits;
  ZeroMemory(&bmi.bmiHeader,sizeof(BITMAPINFOHEADER));
  bmi.bmiHeader.biSize         =sizeof(BITMAPINFOHEADER);
  bmi.bmiHeader.biWidth        =bitmapWidth;
  bmi.bmiHeader.biHeight       =bitmapHeight;
  bmi.bmiHeader.biPlanes       =1;
  bmi.bmiHeader.biBitCount     =24;
  bmi.bmiHeader.biCompression =BI_RGB;
  bmi.bmiHeader.biSizeImage    =0;
  bmi.bmiHeader.biXPelsPerMeter=72;
  bmi.bmiHeader.biYPelsPerMeter=72;
  bmi.bmiHeader.biClrUsed      =0;
  bmi.bmiHeader.biClrImportant=0;

  hbm=CreateDIBSection(NULL, &bmi, DIB_RGB_COLORS,
(void**)&pclrBits, NULL, 0);

  CopyMemory(pclrBits, fullByteArray,widthBytes*bitmapHeight);


  if (fullByteArray != NULL)
  {
    delete [] fullByteArray;
    fullByteArray = NULL;
  }


  //assign the new bitmap handle to the Transform CBitmap
  m_TransformBitmap.m_hObject = hbm;

  //fill in legacy bitmap structure information
  BITMAP bm;
  m_TransformBitmap.GetBitmap(&bm);
  m_bcore.biHeight = bm.bmHeight;
  m_bcore.biWidth = bm.bmWidth;
  m_bcore.biPlanes = bm.bmPlanes;

}
```

Generating a Bitmap from the Complex Data Array (real and imaginary parts)

```
void CRedBoxDoc::GenerateAbsoluteBitmapFromThirdBytes(int
bitmapWidth, int bitmapHeight)
{
    int i;
  float factor, max, min;

  int padwidth = (3*bitmapWidth)%4;
  if(padwidth!=0)
    padwidth = 4-padwidth;
  int widthBytes = 3*bitmapWidth+padwidth;

  BYTE* fullByteArray;
  fullByteArray = new BYTE[widthBytes * bitmapHeight];

  //find max and min points in the bit array
  max = 0;
  min = 255;

  for (i=0; i<bitmapWidth*bitmapHeight; i++)
  {
    if(AbsoluteValue(m_cComplexThirdByteArray[i]) > max)
    {
      max = AbsoluteValue(m_cComplexThirdByteArray[i]);
    }
    else if (AbsoluteValue(m_cComplexThirdByteArray[i]) < min)
    {
      min = AbsoluteValue(m_cComplexThirdByteArray[i]);
    }
  }
  //determine scaling factor
  factor = 255/(float)(max - min);


  //create full bitmap bit array
  int padding, padAmount;
  padding = 0;
  padAmount =  widthBytes - (bitmapWidth * 3);

  int j;

  for (i = 0; i < bitmapHeight; i++)
  {
    for (j=0; j<(bitmapWidth*3); j++)
    {
      fullByteArray[(i*widthBytes)+j] =
(BYTE)((AbsoluteValue(m_cComplexThirdByteArray[(((bitmapHeight-
(i+1))*bitmapWidth*3) + j)/ 3]) - min) * factor) ;
    }
```

```
  }

  //create and populate a bitmap structure and handle
  HBITMAP     hbm;
  BITMAPINFO  bmi;
  BYTE* pclrBits;
  ZeroMemory(&bmi.bmiHeader,sizeof(BITMAPINFOHEADER));
  bmi.bmiHeader.biSize        =sizeof(BITMAPINFOHEADER);
  bmi.bmiHeader.biWidth       =bitmapWidth;
  bmi.bmiHeader.biHeight      =bitmapHeight;
  bmi.bmiHeader.biPlanes      =1;
  bmi.bmiHeader.biBitCount    =24;
  bmi.bmiHeader.biCompression =BI_RGB;
  bmi.bmiHeader.biSizeImage   =0;
  bmi.bmiHeader.biXPelsPerMeter=72;
  bmi.bmiHeader.biYPelsPerMeter=72;
  bmi.bmiHeader.biClrUsed     =0;
  bmi.bmiHeader.biClrImportant=0;

  hbm=CreateDIBSection(NULL, &bmi, DIB_RGB_COLORS,
(void**)&pclrBits, NULL, 0);

  CopyMemory(pclrBits, fullByteArray,widthBytes*bitmapHeight);


  if (fullByteArray != NULL)
  {
    delete [] fullByteArray;
    fullByteArray = NULL;
  }

  //assign the new bitmap handle to the Transform CBitmap
  m_TransformBitmap.m_hObject = hbm;

  //fill in legacy bitmap structure information
  BITMAP bm;
  m_TransformBitmap.GetBitmap(&bm);
  m_bcore.biHeight = bm.bmHeight;
  m_bcore.biWidth = bm.bmWidth;
  m_bcore.biPlanes = bm.bmPlanes;

}
```

Bibliography

Bozic, S.M. (1979). *Digital and Kalman Filtering.* London: Edward Arnold (Publishers) Ltd.

Kain, E. (1999). *The MFC Answer Book.* Reading, Mass.: Addison Wesley Longman Inc.

Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T. (1986). *Numerical Recipes*. New York: Press Syndicate of the University of Cambridge.

Tunaley, J.K.E., Jahans, P.A. and Dixon, J.A. (1989). *Evaluation of Radon Transform Techniques in the Detection and Enhancement of Linear Wake Features in Synthetic Aperture Radar (SAR) Wake Data – Part II.* (Final Report under Contract File No. 003SV.W7714-8-5562/01-SV). London, Ontario: London Research and Development Corporation.

Williams, A.T.E. (2002). *Satellite Radar Detection of Ship Wakes*. (DRDC Ottawa TN2002-079). Ottawa, Ontario: Defence Research and Development Canada.

Toft, Peter. (1996). *The Radon Transform – Theory and Implementation* (Ph.D. Thesis). Lyngby, Denmark: Department of Mathematical Modelling, Technical University of Denmark. (http://eivind.imm.dtu.dk/staff/ptoft/Radon/Radon.html)

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada - Ottawa
3701 Carling Avenue, Ottawa, ON K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document, including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

Analyzing Ship Wakes in Synthetic Aperture Radar Images - Development of the "RedBox" Application (U)

4. AUTHORS (Last name, first name, middle initial)

Higginson, Alan J.

5. DATE OF PUBLICATION (month and year of publication of document)

December 2002

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

77

6b. NO. OF REFS (total cited in document)

3

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

Space Systems and Technology (SST)
Defence R&D Canada - Ottawa
3701 Carling Avenue, Ottawa, ON K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

Project: 15ES11

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC Ottawa TM 2002-157

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

Unlimited Announcement

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

(U)This report summarizes the development of an application for analyzing ships and ship wakes found in satellite SAR imagery. The need for such an application is identified and a previously developed prototype application is examined briefly.
The new application, known as RedBox, offers semi-automated ship wake detection. Ship wakes are detected by identifying significant line-tendencies in the SAR image with the help of a Radon transformation. Image processing techniques such as high-pass filtering using the Fast Fourier Transform are available to enhance the detection process. Current shortcomings of the application are mentioned and future plans for development are discussed. Relevant C++ source code is included for image processing and transformation algorithms.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

(U)Ship wake; Wake extraction; Radon Transform; Wiener Filter;

Defence R&D Canada

Canada's leader in defence
and national security R&D

R & D pour la défense Canada

Chef de file au Canada en R & D
pour la défense et la sécurité nationale

www.drdc-rddc.gc.ca